实验十一　字符输入界面

2020 年秋季学期

杨飞洋 191220138

目录

1. 实验目的：

利用前面实现过的键盘和显示器功能来搭建一个简单的字符输入界面，通过该系统的实现深入理解多个模块之间的交互和接口的设计。

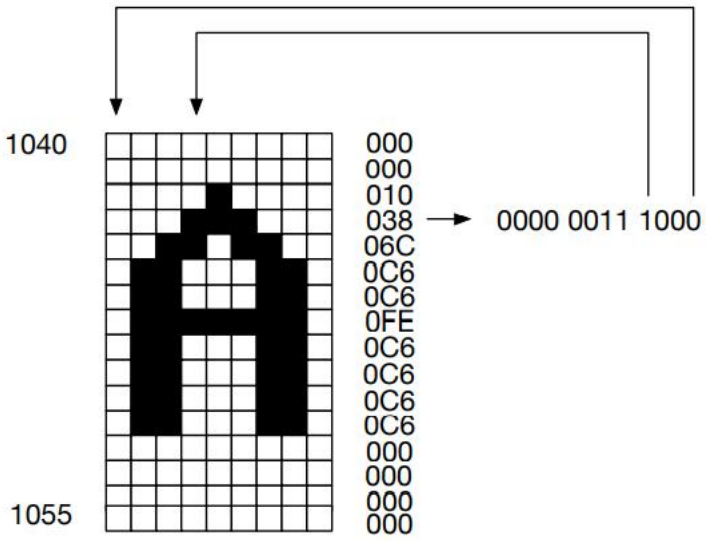
2. 实验原理（知识背景）

字符显示界面只在屏幕上显示 ASCII 字符，其所需的资源比较少。首先，ASCII 字符用 7bit 表示，共 128 个字符。大部分情况下，我们会用 8bit 来表示单个字符，所以一般系统会预留 256 个字符。我们可以在系统中预先存储这 256 个字符的字模点阵。

图 11-1: ASCII 字符字模

这里每个字符高为 16 个点，宽为 9 个点。因此单个字符可以用 16 个 9bit 数来表示，每个 9bit 数代表字符的一行，对应的点为 “1” 时显示白色，为 “0”时显示黑色。

例如，ASCII 字符 “A” 的编码是 41h（十进制 65）。因此其字模对应的地址是 16 × 65 = 1040（文本文件起始从 1 行开始，因此在第 1041 行）。以 A 字符的第 4 行为例，文件中存储的是 038h，二进制对应是 0000 0011 1000。最低位为 0，所以左边第一个像素为 0，左起第 4 到第 6 个像素为 1。如下图所示，此处为方便显示颜色是黑白颠倒的。



有了字符点阵后，系统就不再需要记录屏幕上每个点的颜色信息了，只需 要记录屏幕上显示的 ASCII 字符即可。在显示时，根据当前屏幕位置，确定应 该显示那个字符，再查找对应的字符点阵即可完成显示。对于 640×480 的屏 幕，可以显示 30 行（30×16=480），70 列（70 × 9=630）的 ASCII 字符。系统 的显存只需要 30 × 70 大小，每单元存储 8bit 的 ASCII 字符即可。这样，我们 的字符显存只需要 2.1kByte，加上点阵的 6.144kByte，总共只需要不到 10kByte 的存储，FPGA 片上的存储足够实现了。

3. 实验环境/器材

系统：windows10
开发软件：Quartus 17.1 Lite
开发板：DE10 Standard
仿真环境：ModelSim
芯片：Cyclone V ， 5CSXFC6D6
键盘：ps2 键盘
显示器：vga 显示器

4. 设计思路

基础模块：
通用时钟生成模块：

```
module clkgen(
   input clkin,
   input rst,
   input clken,
   output reg clkout
);
parameter clk_freq=1000;
parameter countlimit=50000000/2/clk_freq; // 自动计算计数次数
reg[31:0] clkcount;
always @ (posedge clkin)
   if(rst)begin
      clkcount=0;
      clkout=1'b0;
   end
   else begin
      if(clken)begin
         clkcount=clkcount+1;
         if(clkcount>=countlimit)begin
            clkcount=32'd0;
            clkout=~clkout;
         end
         else
            clkout=clkout;
      end
      else begin
         clkcount=clkcount;
         clkout=clkout;
      end
   end
endmodule
```

加一行如下的代码就可以生成 25MKHz 的时钟频率

```
clkgen #(25000000) clk_(clk, 1'b0, 1'b1, temp_clk);
```

读取点阵数据模块：
频率不用很高，可以用 my_clock 模块来适当放慢频率
DATA_WIDTH 和 ADDR_WIDTH 分别指代数据长度和地址长度，对应 12*4048 的 txt 文件数据，输入地址，输出相应地址存放的数据

```
module ROM_char
#(parameter DATA_WIDTH=12, parameter ADDR_WIDTH=12)
(
    input [ADDR_WIDTH-1:0] outaddr,
    input clk,
    output reg [DATA_WIDTH-1:0] q
);

    reg [DATA_WIDTH-1:0] ram[2**ADDR_WIDTH-1:0];

    wire clk_1s;

    initial
    begin
        $readmemh("char_table.txt", ram, 0, 4095);
    end

    my_clock clock(clk, clk_1s);

    always @ (posedge clk_1s)
    begin
        q <= ram[outaddr];
    end

endmodule
```

读取键盘数据模块:

```
module keyboard(clk, ps2_clk, ps2_data, sign, temp);
    input clk, ps2_clk, ps2_data;
    output reg sign;

    reg up, upflag1, upflag2, nextdata_n, pre;
    wire myClock, ready, overflow;
    wire [7:0] data, ascii_low, ascii_cap;
    reg [7:0] my_data;
    reg [11:0] vmem_addr = 12'b0;
    reg key, back, enter;
    wire [11:0] row;
    wire [11:0] num;
    wire clear;
    output [8:0]temp;
```

shift 和 caps 的处理之前已经谈过，就不多讲了。主要讲一下 enter 和 backspace 键的处理

```
if (data[7:0] == 8'h66)             //Backspace
begin
    pre = 1;
    key = 0;
    back = 1;
    enter = 0;
    vmem_addr = row<<6+row<<2+row<<1+num;
    my_data = 8'h00;
end
else if (data[7:0] == 8'h5a)        //Enter
begin
    pre = 1;
    key = 0;
    back = 0;
    enter = 1;
    vmem_addr = row<<6+row<<2+row<<1;
    my_data = 8'h00;
end
```

首先是用移位操作刷新数据，再给指示指令 back 和 enter，这两个指令传送到 RAM_backspace 模块中

```
if (back)
begin
   if (ram[count] > 0)
   begin
      ram[count] <= ram[count]-1'b1;
   end
   else if (count > 0)
   begin
      count <= count - 1'b1;
   end
end

if (enter)
begin
   if (count < 29)
   begin
      count <= count + 1'b1;
      ram[count] <= 0;
   end
   else
   begin
      full <= 1;
      for (i=0;i<29;i=i+1)
      begin
         ram[i] <= ram[i+1];
      end
      ram[count] <= 0;
   end
end
```

在此模块中用 count 来指示位置，按照平时的习惯相应地做设置即可。

接下来连接显示器模块:

```
module printchar(clk, reset, clken, vga_clk, hsync, vsync, valid, red, green, blue)
   input clk, reset, clken;
   reg [23:0] data=24'b0;
   output vga_clk, hsync, vsync, valid;
   wire temp_clk;
   wire [9:0] h_addr, v_addr;
   wire [11:0] x;
   wire [11:0] y;
   output [7:0] red, green, blue;
   reg [11:0] ram_addr=12'b0;
   //output reg [8:0] addr;
   wire [7:0]ascii;

   reg [11:0] base, offset_x, offset_y;
   wire [11:0] temp_data;
```

x,y,h_addr,v_addr 相应的指示横纵坐标和行列位置，ram_addr 指示地址信息，temp_data 代表 12 位信息，base 代表起始位，offset_x 和 offset_y 分别指示横纵坐标的偏移量

在 vga_ctrl 模块中相应地加 x,y 变量，

```
//计算当前有效像素坐标
assign h_addr = h_valid ? (x_cnt - 10'd145) : {10{1'b0}};
assign v_addr = v_valid ? (y_cnt - 10'd36) : {10{1'b0}};
assign x = h_addr / 10'd9;
assign y = v_addr >> 4;
```

对应 16*9 的点阵数据

```
always @(posedge clk)
begin
   ram_addr = x << 6 + x << 2 + x << 1 + y;
   //addr=ram_addr[8:0];
   offset_x = h_addr - (x << 3) - x;
   offset_y = v_addr - (y << 4);
   base = {ascii, 4'b0000} + offset_y;
   if (temp_data[offset_x])
      data = 24'hffffff;
   else
      data = 24'b0;
end
```

在 always 语句中计算 offset_x,offset_y,base，用移位操作符和{}操作符可以轻松搞定
顺便指示 data 为黑还是白输送进入 vga_ctrl 模块中

在 read_vmem 模块中用 ram 存储 ascii 码对应的地址数据

```
module read_vmem(clk, count, q);
    input clk;
    input [11:0]count;
    output [7:0]q;

    RAM_videomem vm(8'b0, count, clk, 12'b0, 1'b0, 1'b0, q);

endmodule
```

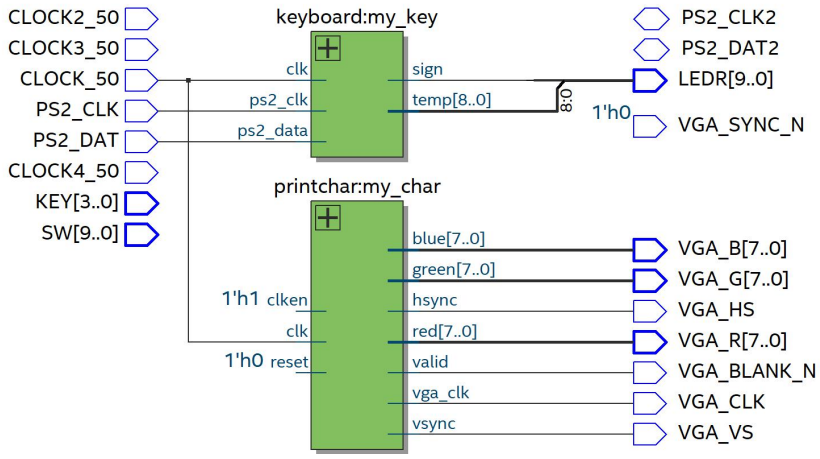
作相应的调用：

```
read_vmem rdvm(temp_clk, ram_addr, ascii);
```

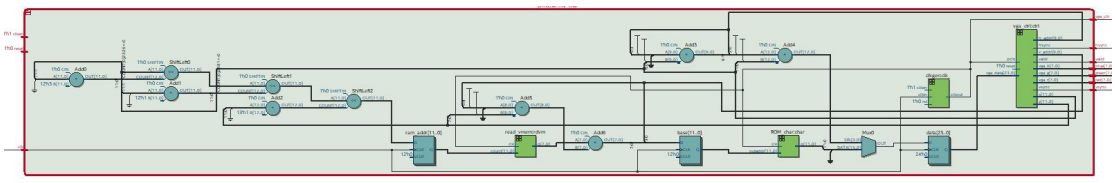
这样整套流程就穿起来了，最后只要在总模块中相应的和输入输出接口对应起来就好了：

```
assign VGA_SYNC_N = 0;

keyboard my_key(CLOCK_50, PS2_CLK, PS2_DAT, LEDR[0], LEDR[9:1]);

printchar my_char(CLOCK_50, 1'b0, 1'b1, VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_N, VGA_R
```

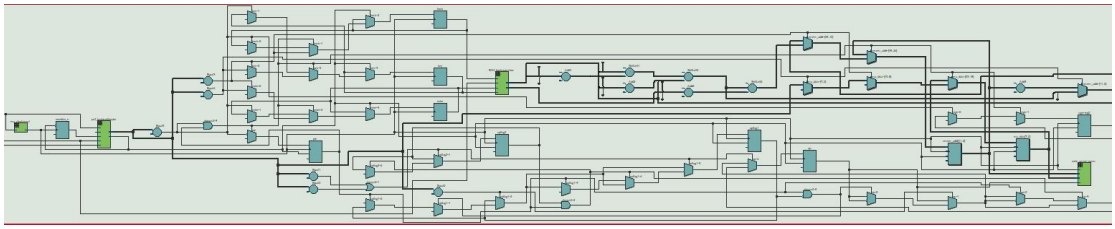
总体的 RTL viewer:



printchar 模块的 RTL viewer:



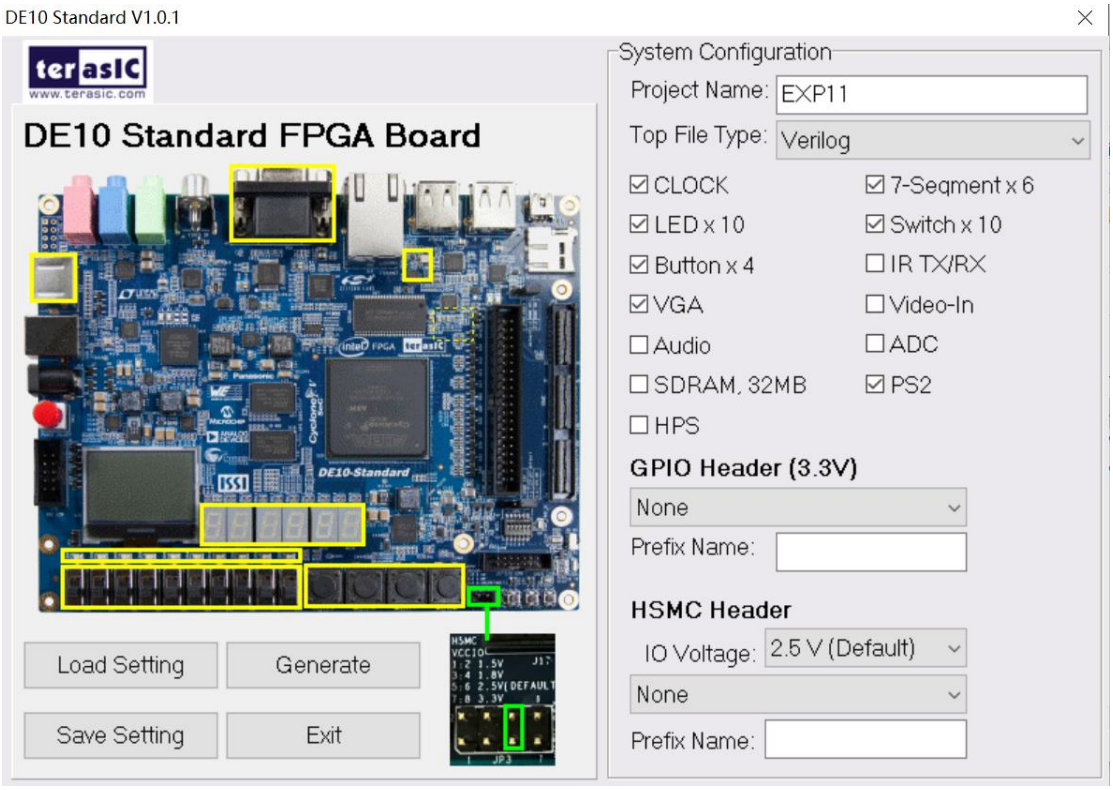
keyboard 模块的 RTL viewer:



5.实验步骤

一．利用 systembuilder 建立工程

选择 PS2 和 VGA，其余常规即可。



二．编写 Verilog 代码

写一下相应模块的代码，在顶层文件中做相应的调用，进行编译调试，直至成功。

三．在板子和键盘上进行测试

6.测试方法

连接键盘和显示器，进行输入，观察输出。

7.实验结果

已验收。

8.实验中遇到的问题

1.一开始对用到的几个时钟频率不大清晰
2.数据的存储方式原本不合理，后来用了 RAM 就好多了。
3.有遇到无数的小 bug，大部分编译信息报错可以直接看出来，其余的通过上网搜索也基本可以解决。

9.实验得到的启示

1．一定要学会用分模块编程，会让结构更加清晰，debug 更加容易
2．有问题可以向老师助教请教、和同学讨论、上网搜，很多时候是思路或结构出现了问题。
3. 有时可能真的是设备的问题。

10.意见和建议

1.这种实验测试文件不好测试，希望老师可以直接说清楚不用测试文件，只需要在板子上测试。
2.希望给一些实验原理的指导，不然自己搞有点懵。
3.感觉这样每次的实验都是接触的全新的东西，又无法深究，这样效率有点低。还是希望深究一点东西。